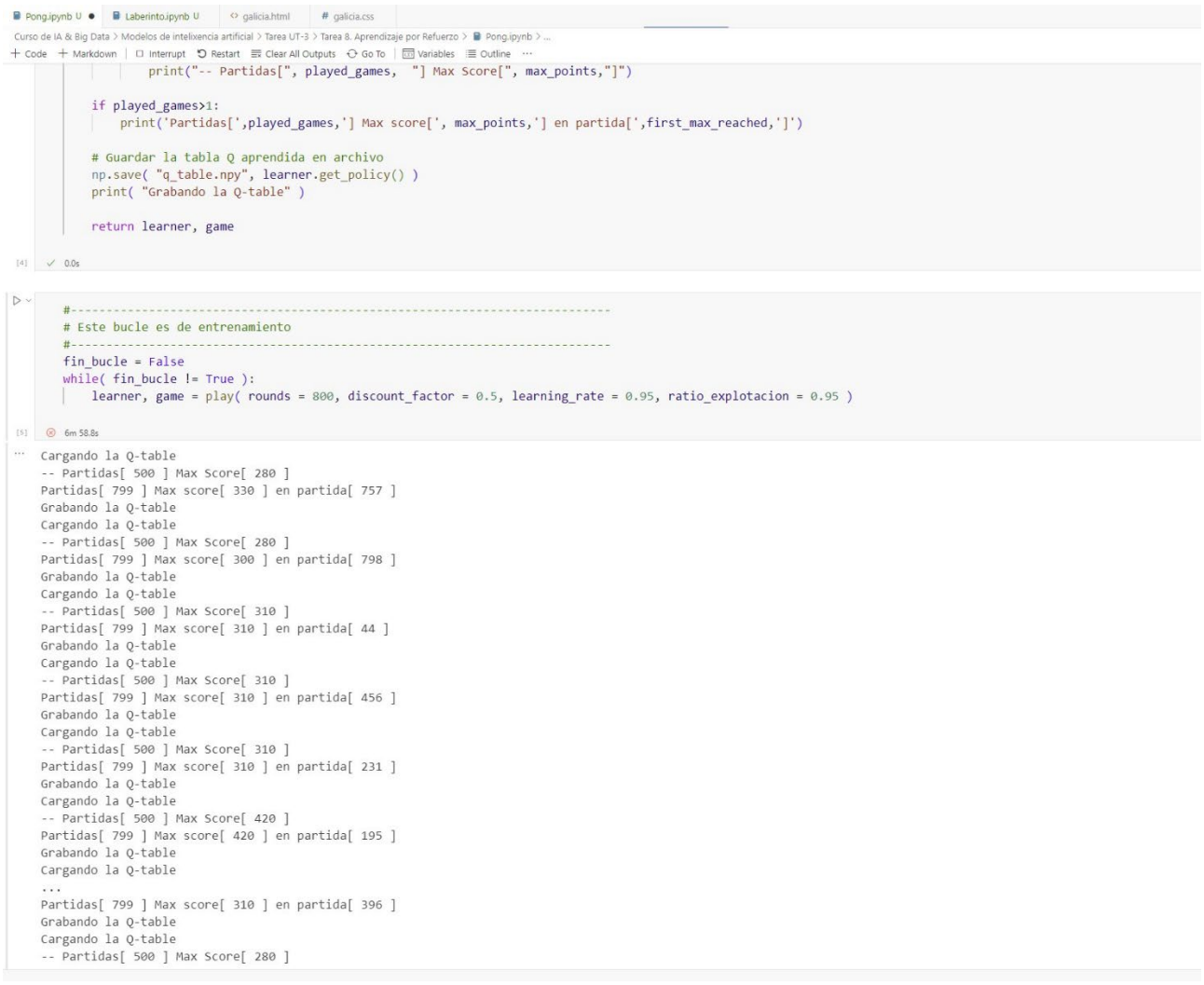
IES de Teis
Avda. de Galicia, 101
36216 – Vigo
886 12 04 64
ies.teis@edu.xunta.es

Busca en Internet información sobre estas cuestiones:

- En el código que tienes a tu disposición del juego Pong, hay un bucle infinito de entrenamiento que cuando quieras lo detienes y después puedes poner a jugar al ordenador y comprobar cuántos puntos obtiene. En la función de entrenamiento puedes modificar los cuatro parámetros de entrada para comprobar si mejora o no el juego.



```
            print("-- Partidas[", played_games,  "] Max Score[", max_points,"]")

        if played_games>1:
            print('Partidas[',played_games,'] Max score[', max_points,'] en partida[',first_max_reached,']')

        # Guardar la tabla Q aprendida en archivo
        np.save( "q_table.npy", learner.get_policy() )
        print( "Grabando la Q-table" )

        return learner, game
```

0.0s

```
#---------------------------------------------------------------------------
# Este bucle es de entrenamiento
#---------------------------------------------------------------------------
fin_bucle = False
while( fin_bucle != True ):
    learner, game = play( rounds = 800, discount_factor = 0.5, learning_rate = 0.95, ratio_explotacion = 0.95 )
```

6m 58.8s

```
Cargando la Q-table
-- Partidas[ 500 ] Max Score[ 280 ]
Partidas[ 799 ] Max score[ 330 ] en partida[ 757 ]
Grabando la Q-table
Cargando la Q-table
-- Partidas[ 500 ] Max Score[ 280 ]
Partidas[ 799 ] Max score[ 300 ] en partida[ 798 ]
Grabando la Q-table
Cargando la Q-table
-- Partidas[ 500 ] Max Score[ 310 ]
Partidas[ 799 ] Max score[ 310 ] en partida[ 44 ]
Grabando la Q-table
Cargando la Q-table
-- Partidas[ 500 ] Max Score[ 310 ]
Partidas[ 799 ] Max score[ 310 ] en partida[ 456 ]
Grabando la Q-table
Cargando la Q-table
-- Partidas[ 500 ] Max Score[ 310 ]
Partidas[ 799 ] Max score[ 310 ] en partida[ 231 ]
Grabando la Q-table
Cargando la Q-table
-- Partidas[ 500 ] Max Score[ 420 ]
Partidas[ 799 ] Max score[ 420 ] en partida[ 195 ]
Grabando la Q-table
Cargando la Q-table
...
Partidas[ 799 ] Max score[ 310 ] en partida[ 396 ]
Grabando la Q-table
Cargando la Q-table
-- Partidas[ 500 ] Max Score[ 280 ]
```

He ido probando distintas configuraciones en los parametros de entrada y esta es la que logra unos mejores resultados, sin llegar a ser una mejora espectacular.

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
886 12 04 64
ies.teis@edu.xunta.es

- Tienes otro código, esta vez de un laberinto, cuyo tamaño y número de obstáculos puedes modificar, como también su tasa de aprendizaje, factor de descuento ante errores, posibilidad de exploración y número de entrenamientos. ¿Has alcanzado algún valor que consideres óptimo o bueno para el programa? ¿qué ocurriría si limitamos drásticamente el número de 'caminos' o intentos?

```
plt.xticks( np.arange( tamano ), [] )
plt.yticks( np.arange( tamaño ), [] )
plt.grid( True, color = 'black' )
plt.show()

#-------------------------------------------------------------------------
# Inicialización de la tabla Q
#-------------------------------------------------------------------------
Q = np.zeros( ( laberinto.shape[ 0 ], laberinto.shape[ 1 ], len( acciones ) ) )

#-------------------------------------------------------------------------
# Parámetros del algoritmo Q-learning
#-------------------------------------------------------------------------
alpha = 0.5     # Tasa de aprendizaje
gamma = 0.4     # Factor de descuento
epsilon = 0.8   # Probabilidad de exploración
caminos = 1000  # Número de iteraciones de entrenamiento

#-------------------------------------------------------------------------
# Entrenamiento del agente
#-------------------------------------------------------------------------
for _ in range( caminos ):
    posicion = ( 1, 1 )  # Estado inicial
    while laberinto[ posicion[ 0 ], posicion[ 1 ] ] != "M":  # Mientras no lleguemos a la salida
        accion = elegir_accion( Q, epsilon, posicion )
        proxima_posicion = ( posicion[ 0 ] + acciones[ accion ][ 0 ], posicion[ 1 ] + acciones[ accion ][ 1 ] )
        if laberinto[ proxima_posicion[ 0 ], proxima_posicion[ 1 ] ] == "#":  # Si la próxima acción es una pared
            proxima_posicion = posicion
        if laberinto[ proxima_posicion[ 0 ], proxima_posicion[ 1 ] ] == "M":  # Si llegamos a la salida
            recompensa = 100  # Recompensa alta por alcanzar la salida
        else:
            recompensa = -1
        Q = actualizar_Q( Q, posicion, accion, recompensa, proxima_posicion, alpha, gamma )
        posicion = proxima_posicion

#-------------------------------------------------------------------------
# Encontramos el camino más corto usando la política aprendida (greedy)
#-------------------------------------------------------------------------
camino_mas_corto = encontrar_camino_mas_corto( Q, laberinto )

#-------------------------------------------------------------------------
# Visualizamos el laberinto con el camino más corto
#-------------------------------------------------------------------------
plt.figure( figsize = ( 8, 8 ) )  # Tamaño de la figura
plt.imshow( np.where( laberinto == '#', 0, 1 ), cmap = 'gray', origin = 'upper' )

#-------------------------------------------------------------------------
# Etiquetamos las posiciones inicial y final
#-------------------------------------------------------------------------
for i, pos in enumerate( camino_mas_corto ):
    if i == 0:
        plt.text( pos[ 1 ], pos[ 0 ], "S", ha = 'center', va = 'center', fontsize = 12, fontweight = 'bold', color = 'red' )
```

En mi experimentación, la conclusión que saco es que valores altos de alpha, gamma y medios o bajos en epsilon son los que tienen un tiempo de ejecucion mas corto.